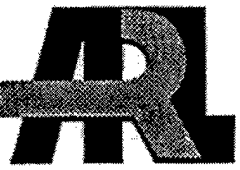
ARMY RESEARCH LABORATORY



A Primary Server for Organizational Identifiers

by Frederick S. Brundick and George W. Hartwig, Jr.

ARL-TR-2530 June 2001

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-2530 **June 2001**

A Primary Server for Organizational Identifiers

Frederick S. Brundick and George W. Hartwig, Jr.
Computational and Information Sciences Directorate, ARL

Abstract

This report describes the design and implementation of an organizational identifier server (OIS). This server is the top level of a hierarchy to assign and maintain a list of unique identifiers for Department of Defense (DOD) organizations. These OrgID numbers are designed to provide a uniform means for digital computers to reference DOD organizations. The OIS accepts requests for OrgIDs from organization servers (OS) and generates sets of unique numbers in response. The OIS also acts as a directory for assigned numbers. OS programs may query the OIS as to who owns a particular OrgID, and the OIS will respond with the name of the OS that was assigned the particular OrgID and its current status.

Acknowledgments

We would like to acknowledge Dr. Sam Chamberlain for creating the original concept of organizational identifiers and serving as a liaison between the authors and the rest of the world. We also wish to thank Dr. Lisa Marvel for taking the time to review this document.

INTENTIONALLY LEFT BLANK.

Table of Contents

INTENTIONALLY LEFT BLANK.

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

With the current push to digitize the Army and ultimately the Department of Defense, one of the problems to be solved is how to identify everyone. Every unit has a name that conveys a large amount of information—such as the echelon, unit type, and who they are assigned to (i.e., *D Company, TF 1-35th Armor*—to human beings). However, these names are difficult for computers to interpret and pass around. It would make sense to identify units with a number that the ubiquitous computer can easily manipulate and exchange with other computers, much like bar codes are used to keep track of physical items. A translation back into military nomenclature could always be done at any time (computers excel at such tasks). Chamberlain [1] has developed a proposal to use unsigned integers as organizational identifiers (OrgID). 32 bits* will allow for 4.3 billion organizations. A problem with using such numbers is how to keep track of them. As proposed by Chamberlain, this bookkeeping is done on two levels and is shown in Figure 1. At the top of this figure is an OrgID Server

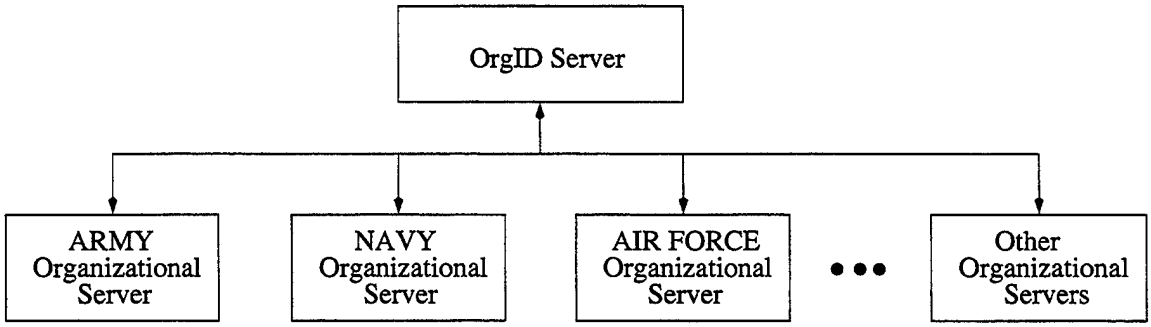


Figure 1. Hierarchy for OrgID Distribution.

(OIS) that serves as the number generator. This program generates the OrgIDs and keeps track of the Organization Server (OS) they are assigned to and their status. The second level is composed of the OSs that keep track of how particular organizations assign the OrgIDs to their various components (i.e., Army Server, Navy Server, etc).

If a user in the field wants to know the owner of a particular OrgID number, he would first query his local OS. It is most likely that this OS would know the unit corresponding to the number in question. If not, the OS program would then query the OIS. The OIS would respond with the OS that had been assigned the number and indicate whether the OrgID was listed as ACTIVE or INACTIVE. The user's OS would then query the named OS. Depending on security considerations and organizational policy, that OS could choose to respond or not. This process is shown graphically in Figure 2. By designing the process in this manner, each OS maintains control of the unit information for which they are responsible.

This report describes the design and implementation of the OIS and the library routines that provide the interface between the OIS and OS. Data replication between multiple OIS machines will be implemented using tools provided by the database vendor and therefore is not discussed.

*32 bits is the size of an unsigned integer on almost all popular computers today.

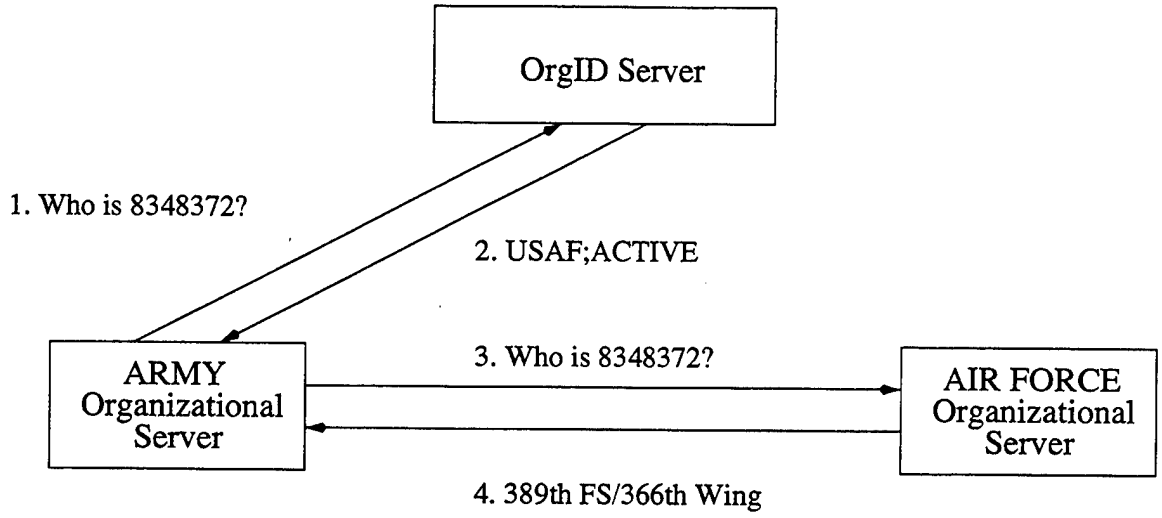


Figure 2. OrgID Query Process.

2. Server Architecture

The OIS is composed of three major components: a relational database management system (RDBMS), a network front end, and a system console. To provide a backup capability, the OIS system is comprised of two OISs running on separate computers (see Figure 3). The

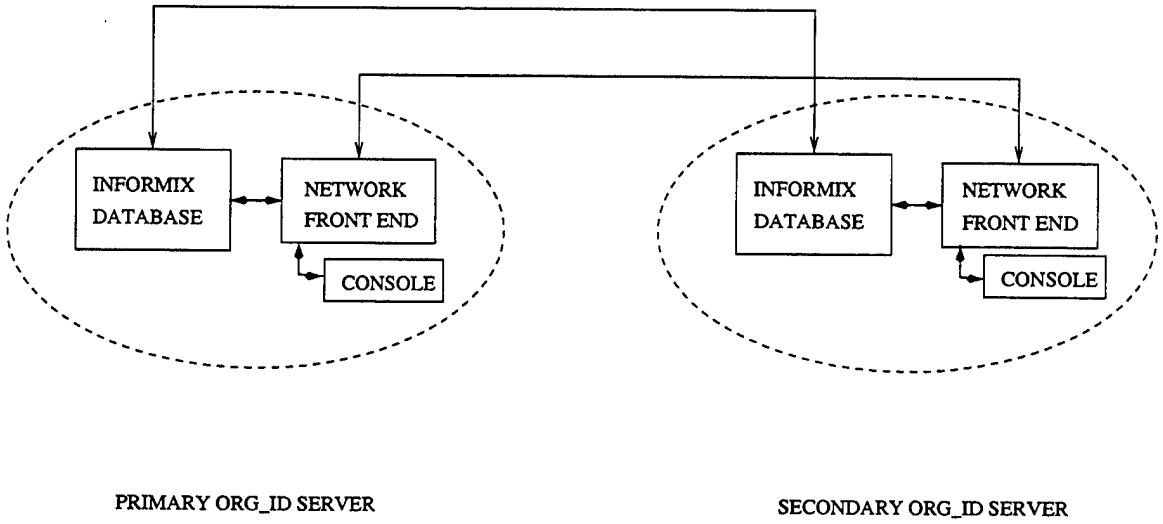


Figure 3. Basic Architecture of the OrgID Server System.

relational database management system selected for this work is Informix. The RDBMS provides a permanent data store and also replicates all OrgID data to a secondary server that is kept in reserve in case problems occur with the primary server.

The network front-end provides communications between the OrgID server and the OS client. It also provides for limited authentication and data security. The console allows a Database Administrator (DBA) to access the program to monitor the state of the database,

create new user accounts, and perform system maintenance operations. To keep the system as flexible and portable as possible, a graphical user interface was not constructed.

3. Data Structures

The data stored by the RDBMS may be partitioned into two parts: (1) the data describing OrgIDs and their use; and (2) data used by the OIS program to keep track of user access information and connection data.

3.1 OrgID Data Tables

OrgID data is maintained in two tables. The first is a table of currently assigned OrgIDs (**orgid**) and is shown in Table 1. The second is a free list of unassigned numbers (**freeblocks**), as shown in Table 2. Both tables are designed to minimize the actual storage required.

Table 1. Assigned OrgID Definition

Name	Type	Definition
idnum	integer	The OrgID number
orgcode	integer	The code number of the OS this number is assigned to
status	char	The current status (Active, Inactive)
reqdate	date	Date this number was assigned to an OS
moddate	date	Date this number was activated/deactivated

Table 2. OrgID Free List Definition

Name	Type	Definition
start	integer	First available number in block
end	integer	Last available number in block

In addition to the actual OrgIDs being used, tables are needed to provide the mapping between organizations and their code numbers (**affiliation**), and information about the users who are authorized to use the OIS (**account**). Tables 3 and 4 show the fields and how they are used.*

A transaction table (**translog**) is maintained to record usage of the OIS, as shown in Table 5. The **recno** field is a simple counter used to provide an audit trail of the exact

*Additional fields could be added to the tables if desired.

Table 3. Organization Information

Name	Type	Definition
orgcode	integer	Code number of the OS
orgname	string	Name of the OS

Table 4. OrgID Account Information

Name	Type	Definition
username	string	Account name
passwd	string	Password for this account
orgcode	integer	Code number of the OS the account is for
address	string	Contact information
phone	string	More contact information
maxoidtrans	integer	Maximum number of requested OrgIDs per transaction
maxoidday	integer	Maximum number of requested OrgIDs per day
maxoiduser	integer	Maximum number of requested OrgIDs for this account

sequence of events. Two of the fields—**success** and **number**—contain different types of values depending on the command. The success codes are returned to the client program via the functions in the application program interface (API) and are explained in Section 4.2.4. Sample data are shown in the Appendix.

Table 5. OrgID Transaction Log

Name	Type	Definition
recno	integer	Transaction number
username	string	Account name
command	char	Command that was issued
success	integer	Success or failure code number
cmddate	date	Date command was given
number	integer	Varies depending on the command

3.2 Table Values

Code values are used in the tables to conserve disk space in accordance with database constraints. The simplest of these are the single-character status codes in the **orgid** table, which are currently 'A' for active and 'I' for inactive. The **translog** table uses both character

codes and numerical codes. These numbers provide more information than a simple pass/fail value, although the main concern is whether or not a command succeeded. The codes used for the commands and the meaning of the number stored in the **number** field are shown in Table 6.

Table 6. Transaction Log Command Codes

Code	Definition	Number
L	Login	UNIX date and time
T	Logout	UNIX date and time
X	Lost Connection	UNIX date and time
R	Request for OrgIDs	Desired number of OrgIDs
A	Change OrgID status to Active	Number of OrgIDs to change
I	Change OrgID status to Inactive	Number of OrgIDs to change
F	Free (delete) OrgID	Number of OrgIDs to free
Q	Query	OrgID that was queried

4. Interface Issues

There are two major interfaces that the OIS must support. The OIS must exchange information with the Informix RDBMS and the various OS client programs.

4.1 The OIS to Informix Interface

Informix supports structured query language (SQL) and a variant developed for use within C programs called embedded SQL for C (ESQL/C). Since the OIS is written in C, it communicates with the Informix RDBMS via ESQL/C. To implement this interface, SQL statements are combined with C source in a file, along with special Informix keywords using SQL syntax. A preprocessor converts this ESQL/C file into a normal C file, with the SQL statements being replaced by calls to C functions that are part of the ESQL/C library. The resulting C file is then compiled by an ordinary C compiler and linked with the libraries supplied by Informix.

The use of ESQL/C combines the power of SQL with the flexibility of C. For example, an SQL SELECT statement normally returns a table of records. With ESQL/C we may write a loop which uses C code to sequentially manipulate each record that is returned from the query. We also have the ability to use conditional flow control statements, such as traditional "if-then-else" constructs, and to perform complex calculations.

4.1.1 Logging In and Out

Since the OIS program runs continuously while waiting for new connections from organization servers, it is not efficient for the OIS to remain connected to the RDBMS because of Informix system overhead. When a new OS client contacts the OIS, the function *iOpenDb* is called, and *iCloseDb* is invoked when the OS logs out. A counter keeps track of the number of connected OS programs. If the counter is zero when *iOpenDb* is called, a connection is made to the RDBMS. Likewise, *iCloseDb* breaks the connection when the counter falls to zero (i.e., the last OS program logs out).

The OIS must validate an OS's credentials before allowing him to submit further requests to the OIS. The function *get_user_info* is given the name of the client attempting to login. It executes a SELECT command on the **account** table to get the client's password, organization, and request limits (limits are detailed in the following section). If the client is found in the table, his information is returned in a data structure; otherwise, a `NULL` value is returned and the connection is terminated.

4.1.2 Getting New OrgIDs

To prevent a user from requesting an inordinately large number of OrgIDs, each user has a transaction limit (see Table 4). There is also a daily limit to prevent a user from trying to bypass the transaction limit by submitting many smaller requests. Finally, a third limit specifies the total number of OrgIDs the user may own. The limits are defined for each account, so everyone does not necessarily have the same limits, and the value of any limit may be set to unlimited. If a limit is exceeded, the appropriate error code is returned.

The function that provides a user with new OrgIDs, *db_get*, is the most complicated in the ESQL/C portion of the OIS. Before it processes the user's request, it first makes sure that the user is within his limits.

A table of available OrgIDs, called **freeblocks**, contains a pair of numbers in each record indicating a range of unused OrgIDs. The function scans the **freeblocks** table (using a SELECT loop) and saves information about each block until it has accumulated the desired number of OrgIDs. It inserts a new record for each OrgID in the **orgid** table, recording the code number of the organization that requested the OrgID, the status (inactive), and the current date. The **freeblocks** table is also updated. If all the numbers in a block are used, that block's record is deleted; if part of a block is used, the starting value of the block is updated to reflect the reduced block size.

After both database tables have been updated, the OrgIDs are returned as a string of number ranges to be passed on to the user.

4.1.3 Changing the Status of OrgIDs

The OIS retains information about the OrgIDs that have been given out. When an organization assigns its OrgIDs, it notifies the OIS and reports the OrgIDs that are now in

use. This is handled by the function *db_set_status*. The current codes are 'A' to activate the OrgIDs, 'I' to deactivate them, and 'F' to free them.

An OrgID may be deactivated. One such case occurs when it has been assigned to a temporary unit such as a joint task force. When the OrgID is unassigned, the OIS must be notified that the number is now inactive. The OS may later reassign the OrgID to another unit, at which time it would tell the OIS to activate the number again.

If an organization decides it no longer needs various OrgIDs, it may free them. This is much more than merely deactivating the numbers. The organization is giving up the ownership of the OrgIDs, and they are deleted from the **orgid** table and returned to the **freeblocks** table.

Before the OIS changes the status of a group of OrgIDs, it first makes sure they are all in the **orgid** table (i.e., they have been assigned) and that they belong to the organization which submitted the command. If any OrgID fails to meet these conditions, the entire command is ignored and an error code is returned. The date the status was modified is recorded in the **moddate** field in the **orgid** table. The only exception is when the OrgID is freed, because in that case the record is deleted from the table. It is not an error to attempt to change the status of an OrgID to its current status. In other words, if the user asks for an active OrgID's status to be set to "active," the command will be executed, even though only the **moddate** field will be changed.

4.1.4 Obtaining Information About an OrgID

A user may wish to obtain information about a particular OrgID. The user may submit that OrgID to the OIS and receive the OS that owns the OrgID and its current status. The function *db_get_owner* runs a SELECT on the **orgid** table, returning an error code if the OrgID has not been assigned.

4.1.5 Miscellaneous Functions

Every command submitted to the OIS is recorded in the table **translog** (see Table 5), regardless of the command's success or failure. The transaction record is stored by the function *record_transaction*. The command codes and other details are explained in Section 3.1, and the success codes are enumerated in Section 4.2.4.

Two functions exist for debugging purposes, *iShowFree* and *iShowCount*. The function *iShowFree* prints the **freeblocks** table. Summary information for the **orgid** table is generated by *iShowCount*, which prints the number of OrgIDs owned by each organization, separated into active and inactive.

4.2 The OIS to OS Interface

The OIS to OS interface has several parts. The PKG software handles the details of the network connections. Above this are a set of OIS routines that manage PKG input and output, and finally there is a library of functions that comprises the API and are called directly by the OS programs.

To assure portability, versions of the PKG, DES, and API libraries have been compiled and executed on the following machine/OS combinations: Sun Ultra 1/Solaris 2.6 and 2.7, Windows NT, and Windows 98. Under Solaris and Windows NT, the compilation was performed by the GNU compiler, gcc. Microsoft's Visual C++ compiler was used for the Windows 98 version.

4.2.1 The PKG Library

The PKG software [2, 3] was developed by Mike Muuss, Charles Kennedy, and Phillip Dykstra at BRL.* It implements a message passing interface for managing client/server communications using TCP/IP connections. Server actions include initialization, client connection, and client request servicing. Client actions include making the server request, reply processing, possible connection negotiation, and data transfer processing. Security is not considered within the PKG routines.

Once the connection is established, it may be used for either full-duplex asynchronous or for synchronous query/response transactions. These transactions are multiplexed on the server side within the PKG interface. Message multiplexing is handled by prefixing a header used to relate client requests to server responses. At the receiving end, the user messages are passed to user-defined routines that provide the PKG-user interface. The calling of these user-defined routines is specified in a user-supplied table. The structure of this table is

```
struct pkg_switch {
     unsigned short pks_type;    /* Type code */
     void (*pks_handler)();      /* Message Handler */
     char *pks_title;            /* Description of message type */
};
```

and the actual table used in the OIS is

```
struct pkg_switch pkg_switch[] = {
     { OIS_ERROR, ois_unk, "Error Message" },
     { OIS_LOGIN, ois_login, "Login Message" },
     { OIS_REQ, ois_req, "Id_req Message" },
     { OIS_RESP, ois_resp, "Id Response" },
     { OIS_LOGOUT,ois_logout, "Logout Message" }
};
```

*The Ballistic Research Laboratory that was incorporated into the Army Research Laboratory in 1992.

The first entry on each line is a PKG message identifier. The second is a pointer to a function that will be called when a message of that type is received; the third is a string used for identification purposes in error messages and data logging. The functions referenced in this table are described in more detail in the next section.

4.2.2 PKG Interface Definition

The interface to PKG software in the OIS program is via the following calls

```
struct pkg_conn *pkg_open ( host, service, protocol, uname, passwd,
        pkg_switch, errlog )
      char *host              /* Name of host to connect to */
      char *service           /* The port to use for this connection */
      char *protocol          /* The protocol to use, TCP always */
      char *uname             /* A username, always ignored here */
      char *passwd            /* A user password, always ignored here */
      struct pkg_switch *pkg_switch     /* The structure relating
                                 message types and handler routines */
      void (*errlog)()        /* A filename for logging errors */
      returns:  pointer to a pkg_conn structure
```

Pkg_open is called by a client to make a connection to a server running on the named host. The return value is the connection handle. Protocol, username, and password are optional, with username and password not being used.

```
void pkg_close ( ptr_to_conn )
      struct pkg_conn *ptr_to_conn      /* The connection to close */
      returns:  none
```

Pkg_close is called by either the client or server to gracefully close a connection.

```
int pkg_permserver ( service, protocol, backlog )
      char *service           /* The port to use for this connection */
      char *protocol          /* The protocol to use, TCP always */
      int backlog             /* Passed to listen(3n) maximum pending
                                 connections queue */
      returns:  file descriptor
```

Pkg_permserver is called by the server to set the service port and to listen for data on the port that is bound to the socket. The return value is the file descriptor of the socket.

```
struct pkg_conn *pkg_getclient ( listen_file_desc, pkg_switch,
        errlog, nodelay_flag )
      int listen_file_desc    /* File descriptor for connections */
      struct pkg_switch pkg_switch      /* The structure relating
                                 message types and handler routines */
      void (*errlog)()        /* Pointer to error logging routine */
      int nodelay_flag        /* Flag to use polling or blocking */
      returns:  pointer to the pkg_connection block, NULL or ERROR.
```

Pkg_getclient is used to accept connections from clients. If possible, the connection request from the client is accepted, a pkg_connection block is created, and a pointer to it is returned to the server. The server may request nonblocked service by setting the nodelay_flag to TRUE.

```
int pkg_get ( ptr_to_conn )
      struct pkg_conn *ptr_to_conn       /* The connection to get from */
      returns:  message_arrived, more_data_coming or ERROR.
```

Pkg_get waits until a message header is received and calls the user-specified message handler for that message type. This routine should be called by a program whenever there is data waiting on a connection and the program is not otherwise waiting on a specific message type. The routine will return directly to the caller if no header is available, or if only a partial message is received; otherwise, it calls the user-specified message type handler.

```
int pkg_send ( message_type, data_buff, buff_len, ptr_to_conn )
      int message_type        /* Type of message from pkg_switch */
      char *data_buff         /* The data to be sent */
      int buff_len            /* The number of bytes to be sent */
      struct pkg_conn *ptr_to_conn       /* The client to send to */
      returns:  number of bytes sent or ERROR
```

Pkg_send constructs a message header for the data buffer and transmits it on the connection. If only part of the message can be sent, the actual number of bytes transmitted is returned. Any data in the stream buffer is first flushed. If the stream buffer needs flushing, the message is less than MAXQLEN (currently 512) bytes long, and sufficient room is left in the stream buffer, this message gets "piggybacked" on by copying it to the buffer before flushing. If available, *writev* is used to send the header and data with one system call.

```
char *pkg_bwaitfor ( message_type, ptr_to_conn )
      int message_type        /* Type of message from pkg_switch */
      struct pkg_conn *ptr_to_conn       /* Connection to wait on */
      returns:  ptr_to_buffer or ERROR
```

Pkg_bwaitfor does a blocking read on the connection until a message of message_type is received. Asynchronous messages and messages of other types are processed while this routine waits. The message is returned in a newly allocated buffer that the caller must free.

4.2.3 The PKG Handling Routines

On the server side, these routines interact with the PKG routines described previously and take care of details such as encryption and message logging. Since they are called from within the PKG software, they have no return values. Proc_incoming is not a message handling routine, but calls the PKG routines to process pending data.

```
void drop_client ( i )
     int i                    /* Index into list of clients */
     returns:  none
```

This routine closes the connection to a client by cleaning this entry in the ACTIVE_CONN array and calling pkg_close.

```
void ois_do_login ( pc, buf )
     struct pkg_conn *pc      /* A pointer to the current
                                 pkg_conn structure */
     unsigned char *buf       /* Buffer containing the assembled
                                 message.  Length and type of message may
                                 be determined from the pkg_conn structure
                                 ( pc->pkc_len, pc->pkc_type ) */
     returns:  none
```

After the PKG connection is made, this routine is invoked by a `LOGIN` message. It handles the password look up and initializes everything that needs it. (NOTE: We may need to drop back and remove `LOGIN` and `LOGOUT` as special message types if they take too long to execute.)

```
void ois_logout ( pc, buf )
     struct pkg_conn *pc      /* A pointer to the current
                                 pkg_conn structure */
     unsigned char *buf       /* Buffer containing the assembled
                                 message.  Length and type of message may
                                 be determined from the pkg_conn structure
                                 ( pc->pkc_len, pc->pkc_type ) */
     returns:  none
```

This PKG switch routine is called when a LOGOUT type of message is received. It frees the ACTIVE_CONN structure, drops the PKG link, and logs the fact. See LOGIN comments for possible future action.

```
void ois_req ( pc, buf )
     struct pkg_conn *pc     /* A pointer to the current
                                pkg_conn structure */
     unsigned char *buf      /* Buffer containing the assembled
                                message.  Length and type of message may
                                be determined from the pkg_conn structure
                                ( pc->pkc_len, pc->pkc_type ) */
     returns:  none
```

This handler routine queues incoming client requests for processing at a later time.

```
void ois_resp ( pc, buf )
     struct pkg_conn *pc     /* A pointer to the current
                                pkg_conn structure */
     unsigned char *buf      /* Buffer containing the assembled
                                message.  Length and type of message may
                                be determined from the pkg_conn structure
                                ( pc->pkc_len, pc->pkc_type ) */
     returns:  none
```

This routine is used on the client side to accept responses to OIS_REQ messages. It should never be called on the server side.

```
void ois_unk ( pc, buf )
     struct pkg_conn *pc     /* A pointer to the current
                                pkg_conn structure */
     unsigned char *buf      /* Buffer containing the assembled
                                message.  Length and type of message may
                                be determined from the pkg_conn structure
                                ( pc->pkc_len, pc->pkc_type ) */
     returns:  none
```

This routine is called by the PKG software when a message of unknown type is received. It simply logs the fact and throws away the buffer.

```
int proc_incoming ( fd )
     int fd                  /* File descriptor for this connection,
                                also index into ACTIVE_CONN array */
     returns:
        < 0 for errors
        0 no errors no messages
        > 0 number of messages processed
```

This function reads and processes any data pending on the furnished file descriptor. It returns less than 0 if it was not given a valid file descriptor for the package connection, or if any part of the process fails. It returns 0 if there was data but not a complete message. Return values greater than 0 are the number of messages processed.

4.2.4 The API Library Functions

The API library of functions is supplied to the Organization Server developer to perform the details of communication with the OIS and to provide a set of functions allowing the OS to interact with the OIS. Currently the following actions are defined:

```
int ois_login ( host_p, host_s, uname, password )
     char *host_p;          /* Name of OrgID Server primary host */
     char *host_s;          /* Name of OrgID Server secondary host */
     char *uname;           /* Name of OS or other approved user */
     char *password;        /* Approved passwd */
     returns:
        BAD_ARGUMENT - One of the input arguments didn't
        make sense.  Generally this is a NULL pointer
        or a variable being out of reasonable range.
        PKG_SEND_FAILURE - PKG software failed for the
        specified connection
        PKG_WAIT_FAILURE - PKG software failed to get an
        expected response on the open connection
        LOGIN_OK_P - Login to the primary server was successful
        LOGIN_OK_S - Login to the primary server failed but
        a successful login to the secondary server was completed
        PKG_CONNECT_FAIL - PKG software failed to connect
        to the specified host
        LOGIN_FAIL - PKG connection was established but
        login failed for an unknown reason
        BAD_USER_NAME - PKG connection was established but
        login failed because of an unrecognized username
        BAD_USER_PW - PKG connection was established but
        login failed because of an incorrect password
```

This function creates the connection to the OIS and attempts to log in using the approved username and password. Return values indicate the status of the connection: PKG_FAIL, LOGIN_FAIL, LOGIN_COMPLETE.

```
int num_req ( req_num, ids )
      unsigned int req_num;  /* The number of OrgIDs requested */
      unsigned int *ids;     /* A pointer to sufficient space to
                                store the requested number of OrgIDs */
      returns:
         BAD_ARGUMENT - One of the input arguments didn't
         make sense.  Generally this is a pointer being NULL
         or a variable being out of reasonable range.
         PKG_SEND_FAILURE - PKG software failed for the
         specified connection
         PKG_WAIT_FAILURE - PKG software failed to get an
         expected response on the open connection
         OVER_RL - Request refused - more OIDs were requested
         than allowed during a single request
         OVER_DL - Request refused - more OIDs were requested
         than allowed during a single day
         OVER_ML - Request refused - more OIDs were requested
         than this OS is permitted to own
         REQ_FAIL - If the request failed for any one of many reasons
         other than the ones mentioned above
         num_ids - If successful, a positive number representing the
         number of OIDs granted is returned
```

This function allows the caller to get a number of OrgIDs. The number requested is compared with the caller's limits. Upon successful completion, this routine returns the value 1, and the requested OrgIDs are placed in the ids array. No order can be assumed in the returned numbers. This routine does not return until a response is obtained.

```
int set_status ( ids, num, status )
      unsigned int *ids;     /* A pointer to the start of num ids */
      int num;               /* Number of entries in the ids array */
      int status;            /* A number indicating status of
                                this number or numbers */
      returns:
         BAD_ARGUMENT - One of the input arguments didn't
         make sense.  Generally this is a pointer being NULL
         or a variable being out of reasonable range.
         PKG_SEND_FAILURE - PKG software failed for the
         specified connection
```

```
        PKG_WAIT_FAILURE - PKG software failed to get an
        expected response on the open connection
        REQ_SUCCESS - Request was successful
        REQ_FAIL - Some problem was encountered and status
        value not changed for any of the OrgID numbers
```

This routine allows the OS program to set the status of OrgIDs assigned to it. Current status values are INACTIVE, ACTIVE, FREE. IDs are initially given a value of INACTIVE when assigned to an OS. The OS must then use this function to indicate those IDs that are issued to units by setting the status to ACTIVE. When the OS no longer has any use for selected IDs, the status may be set to FREE, thereby allowing the OIS to return them to the free list. Numbers less than 8 million cannot be freed since these are pre-assigned to the various OSs. This routine does not return until a response is obtained.

```
int owner_of ( id, owner )
     unsigned int id;        /* An OrgID */
     char *owner;            /* The name of the OS that owns
                                this specified OrgID. */
     returns:
        BAD_ARGUMENT - One of the input arguments didn't
        make sense.  Generally this is a pointer being NULL
        or a variable being out of reasonable range.
        PKG_SEND_FAILURE - PKG software failed for the
        specified connection
        PKG_WAIT_FAILURE - PKG software failed to get an
        expected response on the open connection
        REQ_SUCCESS - Request was successful
        REQ_FAIL - Some problem was encountered or couldn't
        find owner of this OID
```

This function allows the caller to determine the OS that is the owner of the specified OrgID. The result is returned in the string owner that is supplied by the caller. This routine does not return until a response is obtained.

```
int logout ( )
     returns:
        BAD_ARGUMENT - One of the input arguments didn't
        make sense.  Generally this is a pointer being NULL
        or a variable being out of reasonable range.
        PKG_SEND_FAILURE - PKG software failed for the
        specified connection
        PKG_WAIT_FAILURE - PKG software failed to get an
        expected response on the open connection
        LOGOUT_OK - Logout succeeded and connection broken
```

This function terminates the connection to the OIS.

4.2.5 OIS to OS Messages

The API functions generate messages that are sent to the OIS program via the PKG software. PKG network transactions use the TCP protocol, which is a 100% reliable protocol; therefore, no special coding is used to ensure successful transmission. Each user-generated command elicits a response from the OIS. All commands and responses are of the form

```
KEYWORD[arg1;arg2;...]
```

where we have a keyword followed by a '[', and then zero or more arguments separated by ';' and finally ending with a ']'. The details of each command and the expected response are shown below.

COMMAND
```
    LOGIN[uname;password]
```
RESPONSE
```
    LOGIN[status]
      where status may be either ACCEPTED or REFUSED.
```

COMMAND
```
    NUM_REQ[number]
```
RESPONSE
```
    NUM_RESP[number;OID1;OID2;...]
```

COMMAND
```
    SET_STATUS[num;status;OID1;OID2;...]
```
RESPONSE
```
    STATUS_UPDATED[]
```

In the NUM_RESP and SET_STATUS commands, the list of OrgIDs may use the representation OID_1–OID_n if a consecutive set of numbers happens to occur. Note, however, that consecutive numbers are never guaranteed and certainly should not be expected.

COMMAND
```
    OWNER_OF[UID]
```
RESPONSE
```
    OWNER_IS[name;status]
```

COMMAND
```
    LOGOUT[]
```
RESPONSE
```
    GOOD_BY[]
```

5. Security Issues

In any client server system there are many opportunities for attack. Anderson [4] provides representative samples. Most attacks are focused at the endpoints and do not directly attack the encrypted messages passed back and forth. In this section we detail the actions taken in both the OIS program and the application library to implement a security policy. Some basic assumptions include that

- The server machine is secure both in a software and physical sense.
- The network is nonsecure.
- The security of the Organization Server is not the responsibility of the OIS.
- Data at the OIS is relatively nonvaluable compared to data in the OS.

The OIS software is composed of a C program that handles the networking connections with the various OS programs and an Informix RDBMS, and a console interface that will allow the OIS manager to both manage and monitor the OIS. The OIS server software runs on Solaris 2.x UNIX operating systems, while the client libraries run on a variety of operating systems. Attacks on the operating system, OS programs, or the Informix RDBMS are beyond the scope of this project and are not considered further.

The network is clearly the most obvious avenue of attack if, for no other reason, it is open to all the possible threats. Threats can literally be anyone—from hostile countries or terrorists to curious high school students. Fortunately, the OIS operates on low value data. Although OIS is crucial to the entire OrgID concept, the numbers themselves are relatively unimportant. All the OIS program knows is which numbers are assigned to which OS and whether they are active or not. There is no association of numbers with units. Intercepting the data between the OIS and OS is of limited usefulness, so it is sufficient to use a straightforward implementation of the Data Encryption Standard (DES) algorithm [5] to encrypt all transactions between the OIS and OS programs.

The primary threats to the OIS are (1) denial-of-service and (2) a "spoofer" program to falsely manipulate the OIS. If the OS can not obtain OrgIDs from the OIS when needed, then the entire system fails. Such an attack is implemented by either overloading the physical network or flooding the OIS with so many requests that authorized users cannot obtain responses. A "spoofer" is a computer program that pretends to be a valid user. To address the spoofer issues, a need to authenticate clients is required. Security in the client-server model used in the OIS-OS system is provided by passwords and the limited number of legal users. Successful entry into the OIS requires that a previously established password be used. This password is then used to encrypt the entered password itself using the DES algorithms. During the initial login procedure, the username is sent in the clear along with the encrypted password. At the OIS, this password is decrypted using the stored password and they are compared. A successful match constitutes a successful login, a session is established, and all further transactions are encrypted using the user's password.

Smartcard technology was considered, but the overhead required for physical management of the smartcards was considered excessive. In addition, the possible access of the OIS program by allied nations may cause difficulties. Since it is expected that OS programs will connect to the OIS infrequently, it may be difficult to maintain synchronization with the smartcards.

5.1 Network Transaction Security

Existing network and operating system security mechanisms are relied upon to provide a secure base. A DES algorithm is used to encrypt all transactions between the OIS and OS, except for the initial login request.

5.2 Data Storage Security

Data Storage at the OIS site is the responsibility of the Informix RDBMS. All Mission Critical information, such as OrgID tables, usernames, and passwords, will be stored by the RDBMS. These data will then be replicated using the Informix replication scheme to the secondary OIS computer.

6. References

1. Chamberlain, S. C. "Default Operational Representations of Military Organizations," ARL-TR-2172, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, February 2000.
2. Muuss, M. J., P. Dykstra, K. Applin, G. Moss, E. Davisson, P. Stay, and C. Kennedy. "Ballistic Research Laboratory CAD Package, Release 1.21," BRL Internal Publication, U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD, June 1987.
3. Muuss, M. J. "Workstations, Networking, Distributed Graphics, and Parallel Processing." *Computer Graphics Techniques: Theory and Practice*, edited by D. F. Rogers and R. A. Earnshaw, Springer-Verlag, 1990.
4. Anderson, R. J. "Why Cryptosystems Fail." *Communications of the ACM*, vol. 37, no. 11, pp. 32–41, November 1994.
5. Young, E. DES Library. ftp://ftp.psy.uq.oz.au/pub/Crypto/DES/, 1998.

INTENTIONALLY LEFT BLANK.

Appendix:

Sample Transaction Data

INTENTIONALLY LEFT BLANK.

In order to test the OIS, sample data were stored in the various tables and test client programs were run. These clients attempted both successful and failed commands to verify that the OrgID server software worked properly. This discussion is limited to the transaction log because it records a summary of the messages sent to the OIS and the success codes.

The important data from the **account** table are shown in Table A-1. There are four users in three different organizations. Notice that their limits are different.

Table A-1. Sample Account Data

Username	Org	OrgID limits		
ois_test	ARMY	50	1000	-1[a]
client2	NAVY	50	1000	-1[a]
piggy	USMC	5	12	1000
megaman	USMC	10	-1[a]	17

[a]A value of -1 means unlimited.

A portion of an actual transaction log is shown in Table A-2. (See Tables 5 and 6 for details regarding the fields and their contents.) The first two transactions (25 and 26) are both failed login attempts. The user `nobody` did not exist in the **account** table, resulting in a success code of -1, while the user `ois_test` entered an incorrect password.

The third user, `client2`, successfully logged in later the same day. He requested 55 new OrgIDs, but was denied because he exceeded his transaction limit (which was 50). He then asked for and received 5 OrgIDs. While he was logged in, the user `piggy` logged in (transaction 30) and requested some OrgIDs. His first request (for 17 OrgIDs) also exceeded his transaction limit (5). He received 5 OrgIDs in two more transactions, but was denied a third set of 5 because he would have gone over his daily limit (12). He logged out in transaction 35. In a similar test, `megaman` logged in and requested 10 OrgIDs. A second set of 10 was denied because it would have put him over his lifetime limit of OrgIDs (17).

After `megaman` logged out, `client2` submitted some more transactions to the OIS. He successfully freed 2 OrgIDs, activated 2 more, and deactivated an OrgID in transactions 40–42. The next 3 transactions were activation requests that failed. A query of OrgID 3885 returned information, while a query of 39030 failed (the OrgID had not been allocated by anyone). He logged out in transaction 48.

Several days later, `client2` logged in again. Like before, he unsuccessfully requested 55 OrgIDs, then asked for and received 5 OrgIDs. Before he could do anything else, he was disconnected, as shown in transaction 52. Soon after that, the user `ois_test` logged in (with the correct password this time). He successfully submitted several transactions and logged out.

These tests have shown that all requests sent to the OIS are recorded properly. A careful examination of the **orgid** and **freeblocks** tables (not shown here because of their size) confirmed that the actions and results stored in the transaction table are correct.

Table A-2. Actual Transaction Log

recno	username	command	success	cmddate	number
25	nobody	L	-1	08/05/1999	933867435
26	ois_test	L	-2	08/05/1999	933867436
27	client2	L	1	08/05/1999	933873696
28	client2	R	-11	08/05/1999	55
29	client2	R	5	08/05/1999	5
30	piggy	L	1	08/05/1999	933873697
31	piggy	R	-11	08/05/1999	17
32	piggy	R	5	08/05/1999	5
33	piggy	R	5	08/05/1999	5
34	piggy	R	-12	08/05/1999	5
35	piggy	T	1	08/05/1999	933873697
36	megaman	L	1	08/05/1999	933873699
37	megaman	R	10	08/05/1999	10
38	megaman	R	-13	08/05/1999	10
39	megaman	T	1	08/05/1999	933873699
40	client2	F	2	08/05/1999	2
41	client2	A	2	08/05/1999	2
42	client2	I	1	08/05/1999	1
43	client2	A	-1	08/05/1999	3
44	client2	A	-1	08/05/1999	1
45	client2	A	-1	08/05/1999	1
46	client2	Q	0	08/05/1999	3885
47	client2	Q	-1	08/05/1999	39030
48	client2	T	1	08/05/1999	933873707
49	client2	L	1	08/20/1999	935157330
50	client2	R	-11	08/20/1999	55
51	client2	R	5	08/20/1999	5
52	client2	X	1	08/20/1999	935157331
53	ois_test	L	1	08/20/1999	935157336
54	ois_test	R	10	08/20/1999	10
55	ois_test	A	4	08/20/1999	4
56	ois_test	Q	0	08/20/1999	3923
57	ois_test	T	1	08/20/1999	935157337

List of Abbreviations

API	Application Program Interface
ARL	Army Research Laboratory
BRL	Ballistic Research Laboratory
DES	Data Encryption Standard
ESQL	Embedded Structured Query Language
IP	Internet Protocol
OIS	Organization ID Server
OrgID	Organizational Identifier
OS	Organizational Server
RDBMS	Relational Data Base Management System
SQL	Structured Query Language
TCP	Transport Control Protocol

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	OSD OUSD(A&T)/ODDR&E(R) DR R J TREW 3800 DEFENSE PENTAGON WASHINGTON DC 20301-3800
1	COMMANDING GENERAL US ARMY MATERIEL CMD AMCRDA TF 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN 3925 W BRAKER LN STE 400 AUSTIN TX 78759-5316
1	DARPA SPECIAL PROJECTS OFFICE J CARLINI 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	US MILITARY ACADEMY MATH SCI CTR EXCELLENCE MADN MATH MAJ HUBER THAYER HALL WEST POINT NY 10996-1786
1	DIRECTOR US ARMY RESEARCH LAB AMSRL D DR D SMITH 2800 POWDER MILL RD ADELPHI MD 20783-1197

NO. OF COPIES	ORGANIZATION
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CI AI R 2800 POWDER MILL RD ADELPHI MD 20783-1197
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1197
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI IS T 2800 POWDER MILL RD ADELPHI MD 20783-1197
	ABERDEEN PROVING GROUND
2	DIR USARL AMSRL CI LP (BLDG 305)

NO. OF COPIES	ORGANIZATION
1	DUSA OR W HOLLIS 102 ARMY PENTAGON RM 2E660 WASHINGTON DC 20301-0102
3	DIRECTOR RESEARCH OASD C3I RM 3E172 THE PENTAGON WASHINGTON DC 20301-6000
10	HQDA ODISC4 SAIS PAA S RM 1C634 B HABERCAMP 100 ARMY PENTAGON WASHINGTON DC 20310-0107
10	DIRECTOR TPIO ABCS ATZL TP 415 SHERMAN AVE FT LEAVENWORTH KS 66027-2300
5	TSM MCS/GCCS A ATZL TSM USA COMBINED ARMS CENTER 415 SHERMAN AVE UNIT 5 FT LEAVENWORTH KS 66027
2	TSM ASAS USAIC ATZS AS FT HUACHUCA AZ 85613-6000
2	TSM CSSCS USACASCOM ATCL K 2521 A AVE BLDG 11620 FT LEE VA 23801-1701
2	TSM FATDS USAFAS ATSF FSC 3 FT SILL OK 73503-5600
2	TSM FORCE XXI USAAC ATZK XXI FT KNOX KY 40121-5000
2	TSM SHORAD USAADAS ATSA TSM SH FT BLISS TX 79916-3802
2	TSM SOLDIER USAIC&S ATZB TS FT BENNING GA 31905-5405
2	TSM NM USASC ATZH NM FT GORDON GA 30905-5000
2	TSM WIN T USASC ATZH WT FT GORDON GA 30905-5000
2	COMMANDER USAAVNC ATZQ CDO STALLSMITH L AVE BLDG 513 FT RUCKER AL 36362
1	DIRECTOR TPIO SE ATZL NSC TP 410 KEARNEY AVE FT LEAVENWORTH KS 66027-1306
1	DIRECTOR TPIO TD ATSE TPIO USAES FT LEONARD WOOD MO 65473-8929
1	COMMANDER USACAC ATZL CG 415 SHERMAN AVE FT LEAVENWORTH KS 66027-2300
1	HQ TRADOC ATCD ZA FT MONROE VA 23651
1	HQ TRADOC ATCD ZC FT MONROE VA 23651
1	HQ TRADOC ATCD G FT MONROE VA 23651
1	HQ TRADOC ATCD F FT MONROE VA 23651
5	USAFMSA RDD MOFI FMR PT 415 SHERMAN AVE UNIT 7 FT LEAVENWORTH KS 66027-2300
5	USAFMSA RDD LEE MOFI FMR L 700 QUARTERS RD SUITE 349 FT LEE VA 23801-1703

NO. OF COPIES	ORGANIZATION
5	DIRECTOR FDD ATCD F 415 SHERMAN AVE FT LEAVENWORTH KS 66027-2300
5	USAFMSA ADD MOFI FMA 9900 BELVOIR RD SUITE 120 FT BELVOIR VA 22060-5578
5	USAFMSA IMD MOFI ZC IM 401 ARMY PENTAGON WASHINGTON DC 20310-0401
5	USAFMSA HQDA SPPT DIV MOFI ZC SAM 401 ARMY PENTAGON WASHINGTON DC 20310-0401
1	ARMY FORCE MGMT SCHOOL 5500 21ST ST SUITE 1400 FT BELVOIR VA 22060-5923
1	DIRECTOR ARMY FORCE PROGRAMS DAMO FDZ RM 3A522 460 ARMY PENTAGON WASHINGTON DC 20310-0460
2	DIRECTOR C2D AMSEL RD C2 D FORT MONMOUTH NJ 07703
2	US ARMY CERDEC AMSEL RD C2 O SS NIEMELA FORT MONMOUTH NJ 07703
2	US ARMY CERDEC AMSRL ST DD SALIS FORT MONMOUTH NJ 07703
2	PEO C3S SFAE C3S BG BOUTELLE MYER CENTER FT MONMOUTH NJ 07703
5	PEO C3S SFAE C3S HTI CARNEVALE MYER CENTER FT MONMOUTH NJ 07703

NO. OF COPIES	ORGANIZATION
2	PM ATCCS SFAE C3S AT MYER CENTER FT MONMOUTH NJ 07703
5	PM ATCCS CS SFAE C3S AT CS D USECHAK MYER CENTER FT MONMOUTH NJ 07703
2	PM MCS SFAE C3S AT MCS FT MONMOUTH NJ 07703
2	PEO C3S TOC AMDCCS SFAE C3S AD REDSTONE ARSENAL AL 35898
2	PM STCCS SFAE C3S STR 6052 MEADE RD SUITE 101 FORT BELVOIR VA 22060-5260
2	PM INTEL FUSION SFAE C3S INT 1616 ANDERSON RD MCLEAN VA 22102-1616
2	PM FBCB2 SFAE C3S AP BLDG 2525 FT MONMOUTH NJ 07703
1	PM FATDS SFAE C3S FS BLDG 457 FT MONMOUTH NJ 07703-5404
2	PM FATDS SFAE C3S FS TMD SAPHOW BLDG 457 FT MONMOUTH NJ 07703-5404
2	PM WIN T SFAE C3S WIN FORT MONMOUTH NJ 07703

NO. OF COPIES	ORGANIZATION
2	DIRECTOR ADO DAMO ADO LEVINE 400 ARMY PENTAGON WASHINGTON DC 20310-0400
1	COMMANDER USAREUR&7 ARMY AEACG GEN MEIGS APO AE 09014
1	HQ USJFCOM DCINC LTG BURNETTE 1562 MITSCHER AVE NORFOLK VA 23551
5	COMMANDER USAF AC2ISRC CCT BARRINGER 130 ANDREWS ST SUITE 216 LANGLEY AFB VA 23665
5	NAVY WARFARE DEV CMD CODE N341 LCDR ESPE 686 CUSHING RD NEWPORT RI 02841
2	NAVAL POSTGRADUATE SCHOOL CODE SM/HH CDR HATCH 555 DYER RD RM 236 MONTEREY CA 93940-5103
2	SPACE AND NAVAL WARFARE SYSTEMS CENTER LES ANDERSON D4123 53560 HULL ST SAN DIEGO CA 92152-5001
2	C A DEFRANCO JR PE AFRL IFSA 525 BROOKS RD SUITE 1015 ROME NY 13441-4505
2	J SALTON SFAE C3S FIO FT MONMOUTH NJ 07703
2	USA LOG INTEGRATION AGENCY LOIA CD M BLACKMAN 5001 EISENHOWER AVE ALEXANDRIA VIRGINIA 22333-0001

NO. OF COPIES	ORGANIZATION
2	HQDA ODCSLOG DALO PL COL STINE 500 ARMY PENTAGON WASHINGTON DC 20301-0500
4	INST FOR DEFENSE ANALYSIS DR F LOAIZA 1801 N BEAUREGARD ST ALEXANDRIA VA 22311
4	H LAVENDER AMERIND INC 1310 BRADDOCK PL ALEXANDRIA VA 22333
1	COL DAVE MEASELS RET SAIC MS 3 8 1 1710 GOODRIDGE DR MCLEAN VA 22102
2	T CAHILL COMPUTER SCIENCES CORP 1301 VIRGINIA DR 1ST FLOOR FORT WASHINGTON PA 19034
2	COL J BESSLER RET SPRINT MS KSOPAR0301 10895 GRANDVIEW OVERLAND PARK KS 66210
	ABERDEEN PROVING GROUND
18	DIR USARL AMSRL CI DR N RADHAKRISHNAN DR J GANTT AMSRL CI C DR J GOWENS AMSRL CI CT DR J BRAND F BRUNDICK (5 CPS) DR S CHAMBERLAIN G HARTWIG (5 CPS) DR R HELFMAN H INGRAM M LOPEZ

NO. OF
COPIES ORGANIZATION

MEMBERS OF TTCP C3I GROUP TP-10

2 MR JOHN LAWS CHAIRMAN
CIS SYSTEMS DEPT
DEFENCE EVALUATION &
RESEARCH AGENCY
ST ANDREWS RD
MALVERN WORCS WR14 3PS

2 MR JOHN TINDLE UK NATL LDR
CIS SYSTEMS DEPT
DEFENCE EVALUATION &
RESEARCH AGENCY
ST ANDREWS RD
MALVERN WORCS WR14 3PS

2 DR JOHN ROBINSON CA NATL LDR
COMMUNICATIONS RESEARCH
CENTRE CRC CANADA
3701 CARLING AVE
BOX 11490 STATION H
OTTAWA K2H 8S2
CANADA

2 DR ALLAN GIBB CA ARMY MBR
INFORMATION SYSTEMS
TECHNOLOGY SECTION
DREV 2459 PIE XI NORTH BLVD
CP 8800 VAL BELAIR X4683
QUEBEC G3J 1X5
CANADA

2 DR IAIN MACLEOD AU NATL LDR
DSTO C3 RESEARCH CENTER
FERN HILL PARK
DEPT OF DEFENCE
CANBERRA ACT 2600
AUSTRALIA

INTENTIONALLY LEFT BLANK.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2001	Final, February 1999 - August 1999

4. TITLE AND SUBTITLE
A Primary Server for Organizational Identifiers

5. FUNDING NUMBERS
P611102AH48

6. AUTHOR(S)
Frederick S. Brundick and George W. Hartwig, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-CI-CT
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2530

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes the design and implementation of an organizational identifier server (OIS). This server is the top level of a hierarchy to assign and maintain a list of unique identifiers for Department of Defense (DOD) organizations. These OrgID numbers are designed to provide a uniform means for digital computers to reference DOD organizations. The OIS accepts requests for OrgIDs from organization servers (OS) and generates sets of unique numbers in response. The OIS also acts as a directory for assigned numbers. OS programs may query the OIS as to who owns a particular OrgID, and the OIS will respond with the name of the OS that was assigned the particular OrgID and its current status.

14. SUBJECT TERMS	15. NUMBER OF PAGES
database, enterprise key, organizational identifier	34
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-2530 (Brundick) Date of Report June 2001

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name E-mail Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001, APG, MD

POSTAGE WILL BE PAID BY ADDRESSEE

DIRECTOR
U.S. ARMY RESEARCH LABORATORY
ATTN AMSRL CI CT
ABERDEEN PROVING GROUND MD 21005-5067